Computing and Informatics, Vol. 24, 2005, 415–426

GENETIC ALGORITHM FOR SOLVING UNCAPACITATED MULTIPLE ALLOCATION HUB LOCATION PROBLEM*

Jozef Kratica

Mathematical Institute
Serbian Academy of Sciences and Arts
Kneza Mihajla 35/I, pp. 367
11 001 Belgrade, Serbia and Montenegro
e-mail: jkratica@mi.sanu.ac.yu

Zorica Stanimirović, Dušan Tošić, Vladimir Filipović

Faculty of Mathematics, University of Belgrade
Studentski trg 16/IV
11 000 Belgrade, Serbia and Montenegro
e-mail: {zoricast, dtosic, vladaf}@matf.bg.ac.yu

Manuscript received 16 March 2005; revised 12 September 2005
Communicated by Vladimír Kvasnička

Abstract. Hub location problems are widely used for network designing. Many variations of these problems can be found in the literature. In this paper we deal with the uncapacitated multiple allocation hub location problem (UMAHLP). We propose a genetic algorithm (GA) for solving UMAHLP that uses binary encoding and genetic operators adapted to the problem. Overall performance of GA implementation is improved by caching technique. We present the results of our computational experience on standard ORLIB instances with up to 200 nodes. The results show that GA approach quickly reaches all optimal solutions that are known so far and also gives results on some large-scale instances that were unsolved before.

* This research was partially supported by the Serbian Ministry of Science and Ecology under project 144007. The authors are grateful to Andreas Ernst, Sergio García Quiles and Haluk Topcuoglu for useful communication and sending us fixed costs for large-scale UMAHLP instances.

Keywords: Hub location problem, genetic algorithms, evolutionary computation, discrete location and assignment, network design, combinatorial optimization

1 INTRODUCTION

Computer and communication systems, DHL services and postal delivery networks, transportation systems may be observed as hub networks. They include a set of interacting nodes (facilities) with given distance and flow cost between each pair of nodes. The nodes in the network denoted as hub nodes serve as consolidation and connection points between two locations. Each node in the network is assigned to one or more hubs. All of the flow between any pair of nodes can only be realized via specified hubs. Since transportation cost between hubs is lower, consolidating traffic through hub nodes results in lower transportation cost per unit and efficient exploitation of the network.

There are various hub location problems, depending on the imposed constraints in the hub network. For example, number of hubs may be predetermined, capacity restrictions or fixed costs on both hub and/or non-hub nodes may be imposed, etc. Hub location problems can assume one of two allocation schemes:

- single allocation scheme, where each node must be assigned to exactly one hub node. All of flows from/to each node go only via assigned hub;
- multiple allocation scheme, which allows each facility to communicate with more than one hub.

Detailed review of hub location problems and their classification can be found in [5] and [6].

There are several papers in the literature considering UMAHLP. The problem was first formulated in [4]. Dual ascent techniques within a Branch-and-Bound scheme were first applied for solving UMAHLP in [11] on ORLIB ([2]) hub instances with $n \leq 25$ nodes. Similar approach is used in [14], with tighter lower bounds and improved upper bounds. The results are presented on instances with up to 40 nodes.

A new quadratic integer formulation of the problem, based on the idea of multi-commodity network flows was introduced in [1]. This new formulation showed to be suitable for using Branch-and-Bound procedure. The authors present results on their own randomly generated instances with $n \leq 80$ size.

In [3] the mixed integer linear programming (MILP) formulations for three multiple allocation hub location problems, including UMAHLP, are used. For each problem preprocessing procedure and tightening constraints were developed. This approach was tested on standard hub data set including up to 50 nodes.

The main idea in paper [13] was to tighten MILP formulation and reduce the number of constraints using results from the field of the polyhedral structure of set packing problem. Another idea presented in [7] was to consider the dual problem of a MILP formulation. The authors first construct a heuristic method, based on a dual-ascent technique, which produces almost 70 % optimal solutions on ORLIB instances with up to 120 nodes. Heuristic was later embedded in Branch-and-Bound algorithm, that provides optimal solutions in all cases.

2 MATHEMATICAL FORMULATION

In this paper we consider the uncapacitated multiple allocation hub location problem. In this case, no capacities on the nodes are imposed, the number of hubs is not fixed and each non-hub node may be assigned to more than one hub (multiple allocation scheme). Traffic between origin and destination node can be routed via one or more hubs (switching points). Every hub node is located with certain expenses (fixed cost hub problem). The objective is to choose set of hubs and allocate non-hub nodes to the chosen set, so that the sum of total transportation cost and fixed costs is minimized.

Various formulations of UMAHLP arise in the literature and one mixed integer linear programming formulation [7] is used in this paper.

Consider a set $I = \{1, \ldots, n\}$ of n distinct nodes in the network, where each node represents origin/destination or potential hub location. The distance from node i to node j is C_{ij}, and triangle inequality may be assumed [6]. The demand from location i to j is denoted as W_{ij}. Decision variables y_k and x_{ijkm} are used in the formulation as follows:

$$y_k = \begin{cases} 1 & \text{if a hub is located at node } k, \\ 0 & \text{if not.} \end{cases}$$

x_{ijkm} is the fraction of flow W_{ij} from node i that is collected at hub k, and distributed by hub m to node j.

Each path from demand to destination node consists of three components: transfer from an origin to the first hub, transfer between the hubs and finally distribution from the last hub to the destination location. Parameters χ and δ denote unit costs for collection and distribution, while $1 - \alpha$ represents discount factor for transport between hubs. The fixed cost of establishing hub k ($y_k = 1$) is denoted as f_k. The objective is locating some hub facilities in order to minimize the sum of total flow cost and the total cost of location hubs. Using the notation mentioned above, the problem can be written as:

$$\min \sum_{i,j,k,m} W_{ij} \cdot (\chi \cdot C_{ik} + \alpha \cdot C_{km} + \delta \cdot C_{mj}) \cdot x_{ijkm} + \sum_k f_k \cdot y_k \tag{1}$$

subject to

$$\sum_{k,m} x_{ijkm} = 1 \quad for\ every\ i, j \tag{2}$$

$$\sum_m x_{ijkm} + \sum_{m, m \neq k} x_{ijmk} \leq y_k \quad for\ every\ i, j, k \tag{3}$$

$$y_k \in \{0, 1\} \quad for\ every\ k \tag{4}$$

$$x_{ijkm} \geq 0 \quad for\ every\ i, j, k, m. \tag{5}$$

The objective function (1) minimizes the sum of the origin-hub, hub-hub and hub-destination flow costs multiplied with χ, α and δ factors respectively and the sum of fixed costs for establishing hubs. Constraint (2) specify that all the flow is sent between every pair of nodes, while constraint (3) ensures that flow is only sent via opened hubs. Constraints (4) and (5) reflect binary and/or non-negative representation of decision variables. Note that the fact $x_{ijkm} \leq 1$ is implied by constraint (2), and is omitted.

UMAHLP is known to be NP-hard, with exception of special cases (for example when matrix of flows W_{ij} is sparse) that are solvable in polynomial time. If the set of hubs is fixed, the related subproblem can be polynomially solved using shortest-path algorithm in $O(n^3)$ time ([8]).

3 ACCOMPLISHED GA IMPLEMENTATION

3.1 Representation and Objective Function

The binary encoding of the individuals is used in this GA implementation. Each solution is represented by the binary string of length n. Digit 1 in the genetic code denotes that particular hub is established while 0 shows it is not.

Since users can be assigned only to opened hub facilities, only array y_k is obtained from the genetic code. There are no capacities, so the values of x_{ijkm} can be calculated during the evaluation of objective function.

For fixed set of hubs (y_k) the modified version of the well-known Floyd-Warsall shortest path algorithm described in [8] is used. After finding shortest paths between all pair of nodes, it is simple to evaluate objective function. It is done by summing shortest distances multiplied with flows and corresponding χ, α, and δ parameters, and adding fixed cost f_k of established hubs ($y_k = 1$).

3.2 Genetic Operators

Selection operator chooses the individuals that will produce offsprings in the next generation, according to their fitness. Low fitness-valued individuals have less chance to be selected than high fittness-valued ones. We use an improved tournament selection operator, known as fine-grained tournament selection (FGTS). This selection scheme showed to be successful in cases when it is desirable that the size of tournament group has rational instead of integer values. This operator uses real (rational) parameter F_{tour} which denotes desired average tournament size. The first type is held k_1 times and its size is $[F_{tour} + 1]$. The second type is performed k_2 times with the $[F_{tour}]$ individuals participated. Since the value $F_{tour} = 5.4$ is used in this implementation of FGTS, the corresponding values k_1 and k_2 (for $N = 50$ non-elitist individuals) are 20 and 30, respectively. Running time for FGTS operator is

$O(N \times F_{tour})$. In practice, F_{tour} is considered to be constant (not depending on N) that gives $O(N)$ time complexity. For detailed information about FGTS see [10].

After a pair of parents is selected, crossover operator is applied to them producing two offsprings. The operator we use in this GA implementation is one-point crossover. This operator is performed by exchanging segments of two parents' genetic codes after randomly chosen crossover point. One-point crossover is realized with probability $p_{cross} = 0.85$. It means that approximately 85 % pairs of individuals exchange their genetic material.

Modified simple mutation operator used in this GA concept is performed by changing a randomly selected gene in the genetic code of the individual, with certain mutation rate. It may happen during the GA execution that (almost) all individuals in the population have the same gene on certain position. These genes are called frozen. If the number of frozen genes is l, the search space becomes 2^l times smaller and the possibility of premature convergence increases rapidly. The selection and crossover operators cannot change the bit value of any frozen gene and basic mutation rate is often too small to restore lost subregions of search space. If the basic mutation rate is increased significantly, genetic algorithm becomes random search. For this reason, the mutation rate is increased only on frozen genes. Therefore, in this implementation mutation rate for frozen genes is 2.5 times higher $(1.0/n)$, compared to non-frozen ones $(0.4/n)$.

3.3 Generation Replacement Strategy

The population size is 150 individuals. Steady-state generation replacement with elitist strategy is used. Initial population is randomly generated, providing maximal diversity of genetic material. Since the number of hubs to be located is significantly smaller compared to total number of nodes, the probability of generating them in the genetic codes of individuals in the initial population is set to $3.0/n$. This way we obtain "better" individuals for starting GA.

Two thirds of the population are directly passing in the next generation (elite individuals). Genetic operators are applied on the rest of the population, so that only one third of the population is replaced in every generation. Objective value of every elite individual is calculated only once, and this provides significant time savings.

Duplicated individuals are removed in every generation. Their fitness values are set to zero so that selection operator avoids them to enter the next generation. This is very effective method for saving the diversity of genetic material and keeping the algorithm away from premature convergence. Individuals with the same objective function, but in some cases different genetic codes may dominate in the population. If their codes are similar, GA can lead to local optimum. For that reason, it is useful to limit their appearance to some constant N_{rv} (it is set to 40 in this GA application).

3.4 Caching GA

Run-time performance of GA is optimized by caching technique. The main idea is to avoid computing the same objective function value every time when genetic operators produce an individual with same genetic code. The evaluated function values are stored in hash-queue data structure using Least Recently Used (LRU) caching technique. When the same code is obtained again its objective value is taken from the caching table, that provides time-savings. In this implementation the number of individuals stored in the caching table is limited to constant 5 000. For detailed information about caching GA see [12].

Inst.	Opt_{sol}	GA_{best}	t (sec)	t_{tot} (sec)	*gen*	gap_{avg} (%)	σ_{avg} (%)	*eval*	*cache* (%)
10L	221 032.734	opt	0.003	0.113	503	0.000	0.000	664	97.4
10T	257 558.086	opt	0.001	0.114	501	0.000	0.000	709	97.2
20L	230 385.454	opt	0.007	0.206	504	0.000	0.000	2 547	89.9
20T	266 877.485	opt	0.010	0.204	506	0.000	0.000	2 585	89.8
25L	232 406.746	opt	0.015	0.313	505	0.000	0.000	3 401	86.6
25T	292 032.080	opt	0.014	0.295	506	0.000	0.000	3 483	86.3
40L	237 114.749	opt	0.065	0.833	517	0.000	0.000	5 302	79.6
40T	293 164.836	opt	0.017	0.792	501	0.000	0.000	5 217	79.3
50L	233 905.303	opt	0.072	1.434	510	0.000	0.000	6 650	74.1
50T	296 024.896	opt	0.072	1.339	512	0.000	0.000	6 626	74.3
60L	225 042.310	opt	0.075	2.149	506	0.000	0.000	7 248	71.5
60T	243 416.450	opt	0.130	2.417	516	0.000	0.000	7 568	70.8
70L	229 874.500	opt	0.309	3.691	531	0.000	0.000	8 980	66.3
70T	249 602.845	opt	0.152	3.629	513	0.000	0.000	8 100	68.6
80L	225 166.922	opt	0.809	5.119	565	0.000	0.000	9 613	66.1
80T	268 209.406	opt	0.515	4.992	539	0.000	0.000	9 488	65.0
90L	226 857.465	opt	0.368	6.693	518	0.000	0.000	10 266	60.5
90T	277 417.972	opt	0.424	6.619	522	0.000	0.000	10 017	61.9
100L	235 097.228	opt	1.205	8.381	561	0.000	0.000	10 930	61.2
100T	305 097.949	opt	0.155	7.946	505	0.000	0.000	9 746	61.6
110L	218 661.965	opt	0.557	9.695	517	0.000	0.000	10 022	61.5
110T	223 891.822	opt	1.103	10.731	539	0.000	0.000	10 877	59.8
120L	222 238.922	opt	0.885	12.609	524	0.000	0.000	10 443	60.4
120T	229 581.755	opt	2.343	15.077	564	0.000	0.000	12 188	57.0
130L	-	223 814.109	3.117	21.566	563	0.000	0.000	12 198	56.9
130T	-	230 865.451	2.789	22.765	552	0.000	0.000	12 651	54.4
200L	-	230 204.343	25.202	81.456	667	0.696	1.239	16 374	51.2
200T	-	268 787.633	28.688	93.926	701	0.000	0.000	18 778	46.5

Table 1. GA results on AP instances with $\chi = 3$, $\alpha = 0.75$ and $\delta = 2$

4 COMPUTATIONAL RESULTS

In this section we present results of our GA, tested on a AMD Athlon K7/1.33 GHz with 256 MB of internal memory. The code was written in C programming language. The tests are based on standard ORLIB ([2]) data set AP (Australian Post) which is used for testing larger problems. It is obtained from Australian Post delivery system, containing up to 200 nodes representing post code districts. Smaller size AP instances are generated by aggregating basic AP data set. The distances between cities satisfy triangle inequality, but the traffic (flow) between ordered pair of origin-destination nodes is not symmetric. Fixed costs are included in AP data set, as in [9]. The coefficients χ, δ and α that correspond to flow collection, distribution and transportation between hubs take same values as in [7].

Inst.	Opt_{sol}	GA_{best}	t (sec)	t_{tot} (sec)	*gen*	gap_{avg} (%)	σ_{avg} (%)	*eval*	*cache* (%)
10L	221 032.734	opt	0.003	0.113	503	0.000	0.000	664	97.4
10T	257 558.086	opt	0.001	0.114	501	0.000	0.000	709	97.2
20L	230 385.454	opt	0.007	0.206	504	0.000	0.000	2 547	89.9
20T	266 877.485	opt	0.010	0.204	506	0.000	0.000	2 585	89.8
25L	232 406.746	opt	0.015	0.313	505	0.000	0.000	3 401	86.6
25T	292 032.080	opt	0.014	0.295	506	0.000	0.000	3 483	86.3
40L	237 114.749	opt	0.065	0.833	517	0.000	0.000	5 302	79.6
40T	293 164.836	opt	0.017	0.792	501	0.000	0.000	5 217	79.3
50L	233 905.303	opt	0.072	1.434	510	0.000	0.000	6 650	74.1
50T	296 024.896	opt	0.072	1.339	512	0.000	0.000	6 626	74.3
60L	225 042.310	opt	0.075	2.149	506	0.000	0.000	7 248	71.5
60T	243 416.450	opt	0.130	2.417	516	0.000	0.000	7 568	70.8
70L	229 874.500	opt	0.309	3.691	531	0.000	0.000	8 980	66.3
70T	249 602.845	opt	0.152	3.629	513	0.000	0.000	8 100	68.6
80L	225 166.922	opt	0.809	5.119	565	0.000	0.000	9 613	66.1
80T	268 209.406	opt	0.515	4.992	539	0.000	0.000	9 488	65.0
90L	226 857.465	opt	0.368	6.693	518	0.000	0.000	10 266	60.5
90T	277 417.972	opt	0.424	6.619	522	0.000	0.000	10 017	61.9
100L	235 097.228	opt	1.205	8.381	561	0.000	0.000	10 930	61.2
100T	305 097.949	opt	0.155	7.946	505	0.000	0.000	9 746	61.6
110L	218 661.965	opt	0.557	9.695	517	0.000	0.000	10 022	61.5
110T	223 891.822	opt	1.103	10.731	539	0.000	0.000	10 877	59.8
120L	222 238.922	opt	0.885	12.609	524	0.000	0.000	10 443	60.4
120T	229 581.755	opt	2.343	15.077	564	0.000	0.000	12 188	57.0
130L	-	223 814.109	3.117	21.566	563	0.000	0.000	12 198	56.9
130T	-	230 865.451	2.789	22.765	552	0.000	0.000	12 651	54.4
200L	-	230 204.343	25.202	81.456	667	0.696	1.239	16 374	51.2
200T	-	268 787.633	28.688	93.926	701	0.000	0.000	18 778	46.5

Table 2. GA results on AP instances with $\chi = 3$, $\alpha = 0.75$ and $\delta = 2$

The columns in Tables 1–4 contain the following data (in the presented order):

- dimension of the current AP instance, with L denoting "loose" and T "tight" fixed cost;
- optimal solution (Opt_{sol}), if it is known in advance, otherwise "-" is written;
- the best GA solution (GA_{best}), with mark "opt" in cases when GA reaches optimum for the current instance;
- average time t (in seconds) needed to obtain the best GA value;
- average total time t_{tot} (in seconds) for finishing GA;
- average total number of generations;
- average percentage gap of GA solution with respect to Opt_{sol} or GA_{best};

Inst.	Opt_{sol}	GA_{best}	t (sec)	t_{tot} (sec)	*gen*	gap_{avg} (%)	σ_{avg} (%)	*eval*	*cache* (%)
10L	122 038.940	opt	0.002	0.107	501	0.000	0.000	649	97.4
10T	127 425.939	opt	0.005	0.109	503	0.000	0.000	633	97.5
20L	125 309.816	opt	0.009	0.182	506	0.000	0.000	2 465	90.3
20T	129 079.794	opt	0.004	0.182	501	0.000	0.000	2 431	90.3
25L	126 821.800	opt	0.012	0.254	506	0.000	0.000	3 252	87.2
25T	143 422.390	opt	0.016	0.254	509	0.000	0.000	3 157	87.7
40L	124 994.499	opt	0.075	0.689	527	0.000	0.000	5 097	80.8
40T	140 962.910	opt	0.017	0.674	501	0.000	0.000	4 841	80.8
50L	120 871.926	opt	0.054	1.143	508	0.000	0.000	6 155	75.9
50T	152 294.536	opt	0.024	1.114	501	0.000	0.000	6 021	76.1
60L	112 991.944	opt	0.086	1.708	510	0.000	0.000	6 798	73.5
60T	124 961.384	opt	0.092	2.076	511	0.000	0.000	7 395	71.2
70L	114 595.951	opt	0.238	2.702	527	0.000	0.000	7 950	70.0
70T	134 324.296	opt	0.168	3.162	516	0.000	0.000	8 279	68.1
80L	116 505.953	opt	0.553	3.985	554	0.000	0.000	9 225	66.9
80T	138 970.736	opt	0.300	4.295	523	0.000	0.000	9 041	65.6
90L	115 225.601	opt	0.195	5.149	509	0.000	0.000	9 465	63.0
90T	130 558.600	opt	0.428	5.958	526	0.000	0.000	10 153	61.6
100L	123 822.587	opt	0.714	7.028	540	0.000	0.000	10 595	60.9
100T	143 119.855	opt	0.152	7.351	505	0.000	0.000	9 826	61.3
110L	110 192.705	opt	0.989	8.662	544	0.000	0.000	10 686	60.9
110T	114 895.505	opt	0.642	9.162	524	0.000	0.000	10 511	60.1
120L	111 758.347	opt	2.845	12.624	620	0.000	0.000	12 749	59.0
120T	118 376.769	opt	1.011	11.567	531	0.000	0.000	11 425	57.2
130L	-	115286.957	1.863	18.069	543	0.000	0.000	11 938	56.3
130T	-	119538.946	0.688	17.641	511	0.000	0.000	11 525	55.1
200L	-	120377.895	15.616	69.237	625	0.000	0.000	15 578	50.4
200T	-	133716.442	9.944	67.294	573	0.000	0.000	14 964	48.0

Table 3. GA results on AP instances with $\chi = 1$, $\alpha = 0.1$ and $\delta = 1$

- standard deviation σ of the average gap;
- average number of objective function evaluation (*eval*);
- average percentage of savings (*cache*) obtained by using the caching technique.

On each AP instance GA was run 20 times. The maximal number of generations is set to $N_{gen} = 1\,000$ in this GA implementation. The repetition of best objective function value is limited to constant $N_{rep} = 500$.

As can be seen from Tables 1–4, the proposed GA quickly reaches all known optimal solutions ($n \leq 120$) in $t \leq 3.5$ seconds. For other large-scale instances, for which the optimum is not known, GA obtains solutions in $t \leq 28.7$ seconds. The GA concept cannot prove optimality and adequate finishing criterion that will fine-tune the solution quality does not exist. Therefore, as column t_{tot} in Tables 1–4 shows,

Inst.	Opt_{sol}	GA_{best}	t (sec)	t_{tot} (sec)	*gen*	gap_{avg} (%)	σ_{avg} (%)	*eval*	*cache* (%)
10L	125 591.591	opt	0.003	0.107	501	0.000	0.000	628	97.5
10T	127 425.939	opt	0.002	0.106	503	0.000	0.000	624	97.5
20L	126 058.465	opt	0.004	0.176	501	0.000	0.000	2 307	90.8
20T	129 079.794	opt	0.005	0.184	501	0.000	0.000	2 379	90.6
25L	126 900.890	opt	0.020	0.247	514	0.000	0.000	2 958	88.6
25T	143 422.390	opt	0.016	0.261	509	0.000	0.000	3 132	87.8
40L	125 199.814	opt	0.015	0.630	501	0.000	0.000	4 522	82.1
40T	140 962.910	opt	0.016	0.686	501	0.000	0.000	4 781	81.0
50L	124 917.187	opt	0.024	1.108	501	0.000	0.000	5 851	76.8
50T	152 294.536	opt	0.025	1.132	501	0.000	0.000	5 943	76.4
60L	116 799.121	opt	0.073	1.623	507	0.000	0.000	6 330	75.2
60T	124 961.384	opt	0.090	2.124	510	0.000	0.000	7 368	71.3
70L	120 503.243	opt	0.340	2.742	543	0.000	0.000	7 995	70.7
70T	135 016.621	opt	0.138	3.148	512	0.000	0.000	8 085	68.6
80L	119 405.594	opt	0.100	3.623	504	0.000	0.000	8 290	67.3
80T	138 970.736	opt	0.249	4.374	518	0.000	0.000	8 989	65.5
90L	118 611.695	opt	0.554	5.270	539	0.000	0.000	9 579	64.7
90T	130 558.600	opt	0.410	6.035	524	0.000	0.000	10 061	61.8
100L	125 484.484	opt	0.888	6.890	554	0.005	0.023	10 448	62.4
100T	143 119.855	opt	0.161	7.485	505	0.000	0.000	9 795	61.4
110L	116 255.117	opt	0.906	8.330	541	0.000	0.000	10 338	62.0
110T	121 484.974	opt	1.252	9.516	553	0.000	0.000	11 092	60.1
120L	118 048.051	opt	3.500	12.866	651	0.000	0.000	12 985	60.3
120T	122 850.043	opt	1.274	11.522	541	0.000	0.000	11 377	58.2
130L	-	120 773.444	0.922	16.714	519	0.000	0.000	11 094	57.5
130T	-	126 138.979	0.561	17.268	508	0.000	0.000	11 269	55.9
200L	-	122 401.965	13.764	65.355	614	0.201	0.412	14 873	51.7
200T	-	133 772.797	0.637	58.482	501	0.000	0.000	12 889	48.9

Table 4. GA results on AP instances with $\chi = 1$, $\alpha = 0.5$ and $\delta = 1$

our algorithm runs through additional $t_{tot} - t$ time (until the finishing criterion is satisfied), although it already reached the optimal solution.

The proposed GA approach cannot verify optimality of obtained solutions, but represents significant contribution to existing methods for solving UMAHLP, because it is able to solve large-scale instances unsolved before.

Inst.	Opt_{sol}	GA_{best}	t (sec)	t_{tot} (sec)	*gen*	gap_{avg} (%)	σ_{avg} (%)	*eval*	*cache* (%)
10L	125 591.591	opt	0.001	0.105	501	0.000	0.000	621	97.5
10T	127 425.939	opt	0.001	0.108	503	0.000	0.000	625	97.5
20L	126 058.465	opt	0.003	0.179	501	0.000	0.000	2 300	90.9
20T	129 079.794	opt	0.004	0.184	501	0.000	0.000	2 325	90.8
25L	126 900.890	opt	0.021	0.252	513	0.000	0.000	2 942	88.6
25T	143 422.390	opt	0.017	0.266	508	0.000	0.000	3 115	87.8
40L	125 199.814	opt	0.018	0.642	501	0.000	0.000	4 455	82.3
40T	140 962.910	opt	0.018	0.704	501	0.000	0.000	4 767	81.1
50L	124 917.187	opt	0.023	1.134	501	0.000	0.000	5 820	76.9
50T	152 294.536	opt	0.025	1.154	501	0.000	0.000	5 891	76.6
60L	116 799.121	opt	0.088	1.645	510	0.000	0.000	6 251	75.6
60T	124 961.384	opt	0.090	2.178	509	0.000	0.000	7 313	71.5
70L	121 858.663	opt	0.236	2.683	527	0.000	0.000	7 657	71.1
70T	135 016.621	opt	0.194	3.295	518	0.000	0.000	8 216	68.5
80L	119 405.594	opt	0.101	3.710	504	0.000	0.000	8 303	67.2
80T	138 970.736	opt	0.280	4.540	520	0.000	0.000	9 028	65.5
90L	118 611.695	opt	0.520	5.245	536	0.000	0.000	9 369	65.2
90T	130 558.600	opt	0.323	6.085	517	0.000	0.000	9 887	62.0
100L	125 484.484	opt	1.025	7.044	563	0.000	0.000	10 519	62.8
100T	143 119.855	opt	0.186	7.701	506	0.000	0.000	9 850	61.3
110L	119 007.810	opt	0.864	8.363	538	0.000	0.000	10 221	62.2
110T	122 257.504	opt	0.309	8.712	509	0.000	0.000	10 188	60.2
120L	119 561.474	opt	2.130	11.762	588	0.000	0.000	11 699	60.4
120T	122 850.043	opt	1.188	11.610	537	0.000	0.000	11 243	58.4
130L	-	120 773.444	0.907	16.659	519	0.000	0.000	10 980	57.9
130T	-	126 138.979	0.642	17.569	510	0.000	0.000	11 290	55.9
200L	-	122 401.965	20.330	71.403	675	0.050	0.225	16 167	52.2
200T	-	133 772.797	0.647	59.471	501	0.000	0.000	12 896	48.8

Table 5. GA results on AP instances with $\chi = 1$, $\alpha = 0.9$ and $\delta = 1$

5 CONCLUSIONS

An efficient evolutionary meta-heuristic for solving UMAHLP is presented. Binary representation, mutation with frozen genes, limited number of different individuals with same objective value and caching technique were used. The proposed GA quickly obtains solutions that match optimal ones known in literature. It is also able to solve practical size problems that were out of reach for exact methods.

Further research should be directed to parallelization of genetic algorithm and implementation to multiprocessor systems and applying presented approach to similar hub and other location problems.

REFERENCES

[1] Abdinnour-Helm, S.—Venkataramanan, M. A.: Solution Approaches to Hub Location Problems. Annals of Operations Research, Vol. 78, 1998, pp. 31–50.

[2] Beasley, J. E.: Obtaining Test Problems via Internet. Journal of Global Optimization, Vol. 8, 1996, pp. 429–433. Available on: `http://www.brunel.ac.uk/depts/ma/research/jeb/orlib`.

[3] Boland, N.—Krishnamoorty, M.—Ernst, A. T.—Ebery, J.: Preprocessing and Cutting for Multiple Allocation Hub Location Problems. European Journal of Operational Research, Vol. 155, 2004, pp. 638–653.

[4] Campbell, J. F.: Integer Programming Formulations of Discrete Hub Location Problems. European Journal of Operational Research, Vol. 72, 1994, pp. 387–405.

[5] Campbell, J. F.: Hub Location and the P-Hub Median Problem. Operations Research. Vol. 44, No. 6, 1996, pp. 923–935.

[6] Campbell, J. F.—Ernst A. T.—Krishnamoorty, M.: Hub Location Problems. In: H. Hamacher and Z. Drezner (Eds.): Location Theory: Applications and Theory, Springer-Verlag, Berlin-Heidelberg, 2002, pp. 373–407.

[7] Cánovas, L.—García, S.—Marín, A.: Solving the Uncapacitated Multiple Allocation Hub Location Problem by Means of a Dual-ascent Technique. Working paper No. 1, Departamento de Estadística e Investigación Operativa, University of Murcia, Spain, available on: `http://www.optimization-online.org/DB_FILE/2004/01/812.pdf`, 2004.

[8] Ernst, A. T.—Krishnamoorty, M.: An Exact Solution Approach Based on Shortest-paths for P-Hub Median Problem. INFORMS Journal of Computing, Vol. 10, 1998, pp. 149–162.

[9] Ernst, A. T.—Krishnamoorty, M.: Solution Algorithms for the Capacitated Single Allocation Hub Location Problem. Annals of Operations Research, Vol. 86, 1999, pp. 141–159.

[10] Filipović, V.: Fine-Grained Tournament Selection Operator in Genetic Algorithms. Computing and Informatics, Vol. 22, 2003, No. 2, pp. 143–161.

[11] Klincewicz, J.Ġ.: A Dual Algorithm for the Uncapacitated Hub Location Problem. Location Science, Vol. 4, 1996, No. 3, pp. 173–184.

[12] Kratica, J.: Improving Performances of the Genetic Algorithm by Caching. Computers and Artificial Intelligence, Vol. 18, 1999, No. 3, pp. 271–283.

[13] Marín, A.—Cánovas, L.—Landete, M.: New Formulations for the Uncapacitated Multiple Allocation Hub Location Problem. European Journal of Operational Research, 2005 (to appear).

[14] Mayer, G.—Wagner, B.: HubLocator: An Exact Solution Method for the Multiple Allocation Hub Location Problem. Computers and Operations Research, Vol. 29, 2002, pp. 715–739.

Jozef Kratica received his B. Sc. degrees in mathematics (1988) and computer science (1988), M. Sc. in mathematics (1994) and Ph. D. in computer science (2000) from the University of Belgrade, Faculty of Mathematics. In 2002 he joined the Mathematical Institute as a researcher. As a delegation leader he participated on the International Olympiads in Informatics (IOI '90 Minsk – Belarus, IOI '93 Mendoza – Argentina). His research interests include genetic algorithms (evolutionary computation), parallel and distributed computing and location problems.

Zorica Stanimirović received her B. Sc. degree in mathematics (2000) from the University of Belgrade, Faculty of Mathematics. In 2001 she received a graduate students' award from Serbian Ministry of Science. Since 2000 she is a teaching assistant at the Faculty of Mathematics. In 2004 she received her M. Sc. degree in mathematics. Her M. Sc. thesis is concerning hub and other facility location problems. Her research interests also include genetic algorithms, hub and other location problems and combinatorial optimization.

Dušan Tošić received his B. Sc. degree in mathematics (1972), M. Sc. in mathematics (1977) and Ph. D. in mathematics (1984) from the University of Belgrade, Faculty of Mathematics. Since 1985 he has been professor of computer science at the Faculty of Mathematics His research interests include parallel algorithms, optimization and evolutionary computation, numerical solving of differential equations and teaching computer science.

Vladimir Filipović received his B. Sc. degree in computer science (1993) and M. Sc. in computer science (1998) from the University of Belgrade, Faculty of Mathematics. Since 1994 he has been a teaching assistant at the Faculty of Mathematics. His research interests include genetic algorithms, parallel algorithms and operational research.